PROYECTO IA I

INTEGRANTES

MIGUEL RODRIGUEZ
2110063

MANUEL MEDINA
1943270

JUAN MAJIN
2040177

Escuela de ingeniería de sistemas y computación
Universidad del valle, Cali, Colombia
06 de noviembre 2023

Búsqueda No Informada

Amplitud

El algoritmo de búsqueda en amplitud consiste en explorar todos los nodos adyacentes a un nodo dado, antes de explorar los nodos adyacentes a los nodos adyacentes, y así sucesivamente.

En el código, la lista lista_nodos almacena los nodos que aún no han sido explorados. El método busqueda() del algoritmo de búsqueda en amplitud itera sobre la lista de nodos abiertos y expande el nodo que se encuentra más a la izquierda.

El método expandir_nodo() del algoritmo de búsqueda en amplitud crea un nuevo nodo para cada operador que es aplicable al nodo actual. El nuevo nodo se agrega a la lista de nodos abiertos.

El algoritmo de búsqueda en amplitud termina cuando se encuentra un nodo objetivo o cuando la lista de nodos abiertos está vacía.

En el ejemplo que proporcionaste, el algoritmo de búsqueda en amplitud encuentra una ruta para apagar los dos puntos de fuego en 20 pasos.

La siguiente es una explicación detallada de cómo funciona el código:

- La clase Amplitud hereda de la clase Busqueda, que proporciona las funciones básicas para la búsqueda de un estado objetivo.
- El método __init__() de la clase Amplitud inicializa la lista de nodos abiertos con un único nodo inicial.
- El método busqueda() de la clase Amplitud implementa el algoritmo de búsqueda en amplitud. El método itera sobre la lista de nodos abiertos y expande el nodo que se encuentra más a la izquierda.

Costo Uniforme

El algoritmo de costo uniforme consiste en expandir los nodos en orden creciente de costo, desde el nodo inicial hasta que se encuentre el estado objetivo o se agoten todos los nodos.

En el código, la lista lista_nodos almacena los nodos que aún no han sido explorados. El método busqueda() del algoritmo de costo uniforme itera sobre la lista de nodos abiertos y expande el nodo con menor costo.

El la clase Nodo() del algoritmo de costo uniforme se implementa de la siguiente manera:

```python
class Nodo:

  def __init__(self, estado, padre, operador, profundidad,
entorno,punto_de_fuego_uno,punto_de_fuego_dos ,cubeta_en_mano=False,
cubeta_con_agua=0, capacidad_cubeta=0, fuego_apagado=0,
costo=0,fuego_uno_apagado=False,fuego_dos_apagado=False):
    self.estado = estado # [f, c] ubicación actual del bombero
    self.padre = padre  # Nodo
    self.operador = operador # izquierda, arriba, derecha o abajo
    self.profundidad = profundidad
    self.entorno = entorno # varible para los algoritmos que modifican el
ambiente|1
    self.cubeta_en_mano = cubeta_en_mano # true o false
    self.cubeta_con_agua = cubeta_con_agua # 0, 1, 2
    self.capacidad_cubeta = capacidad_cubeta # 0, 1 o 2
    self.fuego_apagado = fuego_apagado # 0, 1 o 2
    self.fuego_uno_apagado = fuego_uno_apagado
    self.fuego_dos_apagado = fuego_dos_apagado
    self.punto_de_fuego_uno=punto_de_fuego_uno
    self.punto_de_fuego_dos=punto_de_fuego_dos
    self.costo = costo # variable para los algoritmos que requieren costo
```

la clase Nodo() crea un nuevo nodo para cada operador que es aplicable al nodo actual. El nuevo nodo se agrega a la lista de nodos abiertos, ordenados por costo en orden decreciente.

Profundidad Evitando Ciclos

El algoritmo de profundidad evitando ciclos consiste en expandir los nodos en orden decreciente de profundidad, desde el nodo inicial hasta que se encuentre el estado objetivo o se agoten todos los nodos. Para evitar ciclos, el algoritmo utiliza una lista de nodos visitados.

El código funciona de la siguiente manera:

- La clase Profundidad hereda de la clase Busqueda, que proporciona las funciones básicas para la búsqueda de un estado objetivo.
- El método __init__() de la clase Profundidad inicializa la lista de nodos abiertos con un único nodo inicial.
- El método busqueda() de la clase Profundidad implementa el algoritmo de profundidad evitando ciclos. El método itera sobre la lista de nodos abiertos y expande el nodo con mayor profundidad.
- El método expandir_nodo() crea un nuevo nodo para cada operador que es aplicable al nodo actual. El nuevo nodo se agrega a la lista de nodos abiertos, ordenados por profundidad en orden decreciente.
- El método evitar_ciclo() verifica si el estado del nodo hijo ya ha sido visitado. Si es así, el método devuelve False para evitar que el nodo se expanda. De lo contrario, el método devuelve True y el nodo se agrega a la lista de nodos visitados

.
La diferencia entre el algoritmo de profundidad evitando ciclos y el algoritmo de profundidad normal es que el algoritmo de profundidad evitando ciclos utiliza una lista de nodos visitados para evitar que se expandan nodos que ya se han visitado. Esto ayuda a garantizar que el algoritmo encuentre una ruta al estado objetivo, incluso si existen ciclos en el espacio de estados.

Búsqueda Informada

A*

La heurística personalizada calcula la distancia Manhattan entre el estado actual y el punto de fuego que está más cerca de ser apagado. Esto se hace porque el objetivo del problema es apagar los dos puntos de fuego lo más rápido posible.

El código funciona de la siguiente manera:

- La clase A_estrella hereda de la clase Busqueda, que proporciona las funciones básicas para la búsqueda de un estado objetivo.
- El método __init__() de la clase A_estrella inicializa la lista de nodos abiertos con un único nodo inicial.
- El método busqueda() de la clase A_estrella implementa el algoritmo A*. El método itera sobre la lista de nodos abiertos y expande el nodo con menor valor de f.
- El método expandir_nodo() crea un nuevo nodo para cada operador que es aplicable al nodo actual. El nuevo nodo se agrega a la lista de nodos abiertos, ordenados por costo en orden decreciente.
- El método heuristica() calcula la distancia Manhattan entre el estado actual y el punto de fuego que está más cerca de ser apagado.

Avara

El algoritmo de búsqueda avara consiste en expandir el nodo con menor valor de f, donde f es una función que combina una estimación del costo total del camino desde el nodo actual hasta el estado objetivo y la profundidad del nodo actual.

En el problema del bombero, el objetivo es apagar los dos puntos de fuego lo más rápido posible. La heurística utilizada en el código que proporcionaste calcula la distancia Manhattan entre el estado actual y el punto de fuego que está más cerca de ser apagado. Esta distancia es una estimación del costo total del camino desde el nodo actual hasta el estado objetivo, ya que la distancia Manhattan es una medida de la distancia entre dos puntos en un plano.

El código funciona de la siguiente manera:

- La clase Avara hereda de la clase Busqueda, que proporciona las funciones básicas para la búsqueda de un estado objetivo.
- El método __init__() de la clase Avara inicializa la lista de nodos abiertos con un único nodo inicial.
- El método busqueda() de la clase Avara implementa el algoritmo de búsqueda avara. El método itera sobre la lista de nodos abiertos y expande el nodo con menor valor de f.
- El método expandir_nodo() crea un nuevo nodo para cada operador que es aplicable al nodo actual. El nuevo nodo se agrega a la lista de nodos abiertos, ordenados por f en orden decreciente.
- El método heuristica() calcula la distancia Manhattan entre el estado actual y el punto de fuego que está más cerca de ser apagado.

Heurística Utilizada

La heurística utiliza es la de manhattan hacia los dos fuegos y si uno ya esta apagado solo seria uno de los puntos, La heurística de Manhattan es común en problemas de búsqueda, ya que calcula la distancia entre dos puntos en una cuadrícula considerando solo movimientos horizontales y verticales, sin considerar movimientos diagonales. la heurística que calcula la distancia entre los dos puntos de fuego (o uno si uno está apagado), Esta distancia se podría utilizar para guiar la búsqueda, preferentemente hacia los estados que minimicen esa distancia a los puntos de fuego. Los puntos de fuego se almacenan en la lista puntos_fuego. El método fuego_apagado de la clase Nodo indica si un punto de fuego está apagado.

Admisibilidad De La Heurística Planteada

La heurística planteada es admisible porque nunca sobreestima el costo real del camino desde el nodo actual hasta el estado objetivo.

En el problema del bombero, el objetivo es apagar los dos puntos de fuego lo más rápido posible. La heurística planteada calcula la distancia Manhattan entre el estado actual y el punto de fuego que está más cerca de ser apagado. Esta distancia es una estimación del costo real del camino desde el nodo actual hasta el estado objetivo, ya que la distancia Manhattan es una medida de la distancia entre dos puntos en un plano.

Por lo tanto, la heurística planteada nunca sobreestima el costo real del camino desde el nodo actual hasta el estado objetivo. Esto se debe a que la distancia Manhattan es una medida del costo real del camino, y la heurística planteada solo agrega una constante al valor de la distancia Manhattan.

En otras palabras, la heurística planteada cumple con la siguiente condición:

```python
costo_real(nodo_actual, estado_objetivo) <= f(nodo_actual)
```

donde:

- costo_real(nodo_actual, estado_objetivo) es el costo real del camino desde el nodo actual hasta el estado objetivo.
- f(nodo_actual) es el valor de la heurística en el nodo actual.